UBL Methodology for Code List and Value Validation

Working Draft 0.8, 24 November 2006 17:50z - Draft 1

Document identifier:

UBL-codelist-methodology-0.8

Locations:

Persistent version: TBD
Current version: http://www.oasis-open.org/committees/document.php?document_id=21324
Previous version: http://www.oasis-open.org/committees/document.php?document_id=20807

Technical committee:

OASIS Universal Business Language (UBL) TC

Chairs:

Jon Bosak, Sun Microsystems
Tim McGrath <tmcgrath@portcomm.com.au>

Editor:

G. Ken Holman, Crane Softwrights Ltd.

Abstract:

This Working Draft describes a methodology and supporting document types with which trading partners can agree unambiguously on the sets of code lists, identifiers and other enumerated values against which exchanged documents must validate. The illustration context of this specification is the Universal Business Language 2.0 however the methodology is presented in such a way as to apply to any context. This illustration uses genericode files for the external representation of coded values, however the methodology is presented in such a way that any representation of coded values can be used. This methodology is packaged with document models, functional stylesheets and demonstration test files for both Windows and Linux environments that can be executed to demonstrate the behaviors.

Status:

This is a work in progress and does not at this time represent the consensus of the UBL Technical Committee.

Please send comments on this specification to the <ubl-dev@lists.oasis-open.org> list. To subscribe, send an email message to <ubl-dev-request@lists.oasis-open.org> with the word "subscribe" as the body of the message.

Notices:

Table of Contents

Appendix

1. Introduction

This Working Draft describes a methodology and supporting document types with which trading partners can agree unambiguously on the sets of code lists, identifiers and other enumerated values against which exchanged XML documents of any vocabulary must validate.

Note

While this methodology was developed to meet requirements for the OASIS Universal Business Language (UBL) 2.0 [UBL 2.0], it can be applied to any XML document and instances of any XML vocabulary.

Schemata describe the structural and lexical constraints on a document. Some information items in a document are described in the schema lexically as a simple value whereby the value is a code representing an agreed-upon semantic concept. The value used is typically chosen from a set of unique coded values enumerating related concepts. These sets of coded values are sometimes termed *code list*s. Some information items with simple values may represent identifiers, and other items may have values that are constrained by business rules.

For some commonly-understood concepts, publicly-available enumerations of coded values are published and maintained by registration authorities regarded as the custodians of the code list of defined values and their associated concepts. A schema may constrain a value to be one of the entire set of published values so as to ensure the value used represents the published concept. A common example of a publicly-maintained code list is that which enumerates currency value indications [currency] and is used for illustrative purposes in this document.

The use of coded values may also be specified where the coded values themselves are merely agreed upon amongst users but not formally enumerated as a constraint on the definition of a document type. A schema thus constrains the document instance value to be a coded value, but the coded value itself is unconstrained. An example from UBL is `cbc:CountrySubentityCode` that constrains the indication of jurisdictional or administrative boundaries below the country level, such as provinces (as in Canada) and states (as in the United States). UBL schemata do not constrain the coded values used for information items representing this concept.

Trading partners may agree to use the published UBL schemata for constraining the documents exchanged for electronic commerce, but may find the constraints of some code lists therein too loose. For two examples, the schema-expressed enumerated list of currency indications may contain many more items that the parties are willing to use, and the lack of an enumeration of country sub-entity codes might allow nonsensical or undesired values to be used. Thus, the UBL schemata successfully validate an exchanged document against the standardized constraints, but allow information to be represented that is not agreed upon by the parties. Trading partners might, then, wish to constrain the currency indications and country sub-entity indications used in the exchanged documents.

Furthermore, trading partners may wish to agree that different sets of values from the same code lists be allowed at multiple locations within a single document (perhaps allowing the state for the buyer in an order be from a different set of states than that allowed for the seller). Large or published schemata might not be able to accommodate such differentiation very elegantly or robustly, or possibly could not be able to express such varied constraints due to limitations of the schema language's modeling semantics. Moreover it is not necessarily the role of the creators of schemata to accommodate such differentiation mandated by the use of their work products.

Having a methodology and supporting document types with which to perform code list value validation enables parties involved in document exchange to formally describe the sets of coded values that are to be used and the document contexts in which those sets are to be used. Such a formal and unambiguous description can then become part of a trading partner contractual agreement, supported by processes to ensure the agreement is not being breached by a given document instance.

Note

This is *not* the standard for code list schema representation (as in UBL Naming and Design Rules (NDR)), nor is it the standard for external code list coded value enumeration representation (as in genericode [genericode] files), but rather it is only the methodology for value validation given that you have some instances being validated by a schema with agreed-upon values represented in supplemental files.

Both the ZIP [UBL-codelist-methodology-0.8.zip] and TAR/GZ [UBL-codelist-methodology-0.8.tar.gz] compressed packages of the documentation for this methodology each include the stylesheet, data and test files that are referenced in the prose. While these files as delivered are configured for use with UBL and with genericode, they can be adapted for instances of any XML vocabulary and any representation of code lists.

2. Terminology

2.1. Key words

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this Working Draft are to be interpreted as described in [RFC Keywords]. Note that for reasons of style, these words are not capitalized in this document.

2.2. Definitions

business rule	A *business rule* applied to an XML document may be a simple constraint or a co-occurrence constraint on the value an information item as dictated by algorithms or factors outside of the formal document definition.
co-occurrence constraint	A *co-occurrence constraint* is a constraint on an information item that is dependent upon the value of another information item.
document context	Every element and attribute information item below the document element of an XML document is in a *document context* described by its hierarchical ancestry of elements. A fully qualified document context specifies the information item's precise location in the document.
ISO/IEC 19757-3 Schematron	ISO Schematron [Schematron] is an assertion-based schema language used to formally express the constraints and co-occurrence constraints on information items in XML documents. Publicly available implementations of ISO Schematron validating processors include XSLT stylesheets (included in the ZIP file associated with this methodology) and Python programs [Scimitar]. See Section 9.1, “Using ISO/IEC 19757-3 Schematron” for more details.
pattern	The expression of a set of assertion validation rules is encapsulated in ISO Schematron as a *pattern*. There can be any number of patterns in a complete ISO Schematron schema expression, either defined internally or referenced externally.

2.3. Key concepts

code list representation	The representation of a code list is the XML vocabulary with which the codes of a set are enumerated and described, not the abstract list of codes themselves. For example, the international set of currency codes is maintained by the UN/ECE who specifies the code values and their meanings. This abstract set of codes may have many *code list representations*, including database tables, comma-separated-values records, etc. The UBL 2.0

	package includes an XML representation of this set of codes using the genericode XML vocabulary.
naming and design rules	The structural and lexical constraints for the XML document-based representation of information items is described by an XML document schema. *Naming and design rules* govern the synthesis of schema expressions to ensure regularity, consistency and accuracy of the document markup structure model.
value validation	Distinct from structural and lexical validation checking the element and character-level constitution of the representation of an information item, *value validation* is the process where constrained information items are checked for having appropriate values. Value validation is only meaningful after structural and lexical validation has confirmed the value meets its representation constraints.

3. Codes, identifiers and other enumerated values

UBL and other standards provide distinctions between codes, identifiers and other enumerated values. While there are no hard and fast rules for deciding whether an information item is expressed as a code or as an identifier, it is often the case that a code will identify a concept, a group or a type, while an identifier will identify a particular thing or a singleton that is distinct from others of the same concept, group or type. Codes are not typically unique in their application, while identifiers typically are unique.

Codes are thematically grouped in a code list such that users of the coded values can use one or more values only from the group. Thus using code lists, upstream applications can prevent users from entering nonsensical or inapplicable choices of values for a particular information item. Downstream applications can make assumptions about an information item when it is known its value has been pre-validated as being a member of the group. UBL provides for the specification of code list and identifier meta data where the code list properties can be specified to more precisely identify the group in which coded values belong.

Identifiers may be enumerated but often they are synthesized following the rules of a particular scheme, possibly a pattern of characters or the result of an algorithmic calculation. Such identifier values are typically open-ended while codes are often from a closed set of values. UBL provides for the specification of meta data where the scheme by which the identifier value is generated, and the scheme's properties, can be indicated in conjunction with the identifier itself.

These conventions do not preclude a code list from being synthesized as an enumeration, having applied the algorithm of a particular scheme to create a closed set of values. An example from UBL is the very large code list enumerating shipping container sizes, synthesized by combining a number of closed ranges of values into all meaningful permutations of codes. No attempt is made to express the codes algorithmically, nor does the genericode specification provide for such an algorithmic expression.

Nor do these conventions preclude an identifier scheme to be processed in a limited fashion to create a closed set of identifier values from which users must select only acceptable values. An example from UBL could be the account identifiers, where an identifier scheme is used to enumerate the accounts, but the accounts dealt with by trading partners are not open-ended and parties must choose from a pre-determined set of account identifiers.

Of course any information item that is not a code list or an identifier can be expressed as a value that is a member of a predetermined set of values. Trading partners may wish to agree to limit any component of their information interchange based on business requirements for either party.

The UBL methodology for code list and value validation is agnostic to the method by which information items are declared. While there are assumptions made in the delivered version of these support tools that a value's associated meta data is structured as codes or identifiers or otherwise in UBL (see Section 6.1, “ UBL instance meta data”), these files can be changed to meet requirements for other vocabularies.

Trading partners can, therefore, utilize the UBL methodology for code list and value validation as a process in the information management flow to validate the values used in any information item with a constrained set of values, not only code lists.

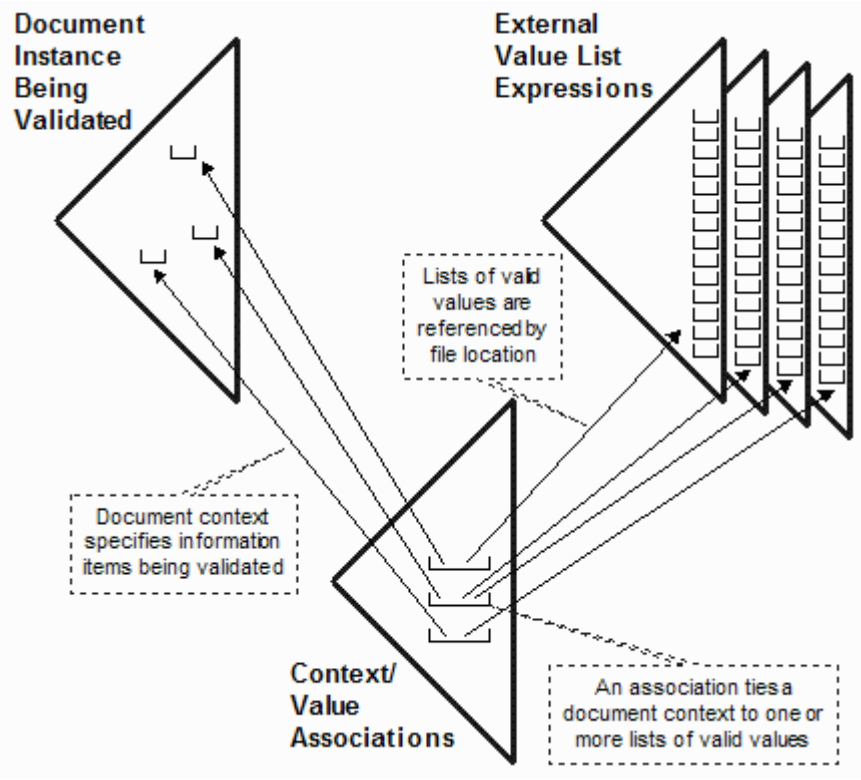
4. Methodology overview and example

4.1. Objectives and process overview

The objective of applying this methodology to a set of document instances being validated is to express the lists of values that are allowed in the contexts of information items found in the instances. One asserts that particular values must be used in particular contexts, and the validation process confirms the assertions do not fail.

Figure 1, “Methodology context association” depicts a file of context/value associations in the lower center, where each association specifies for information items in the document instance being validated which lists of valid values in external value list expressions are to be used.

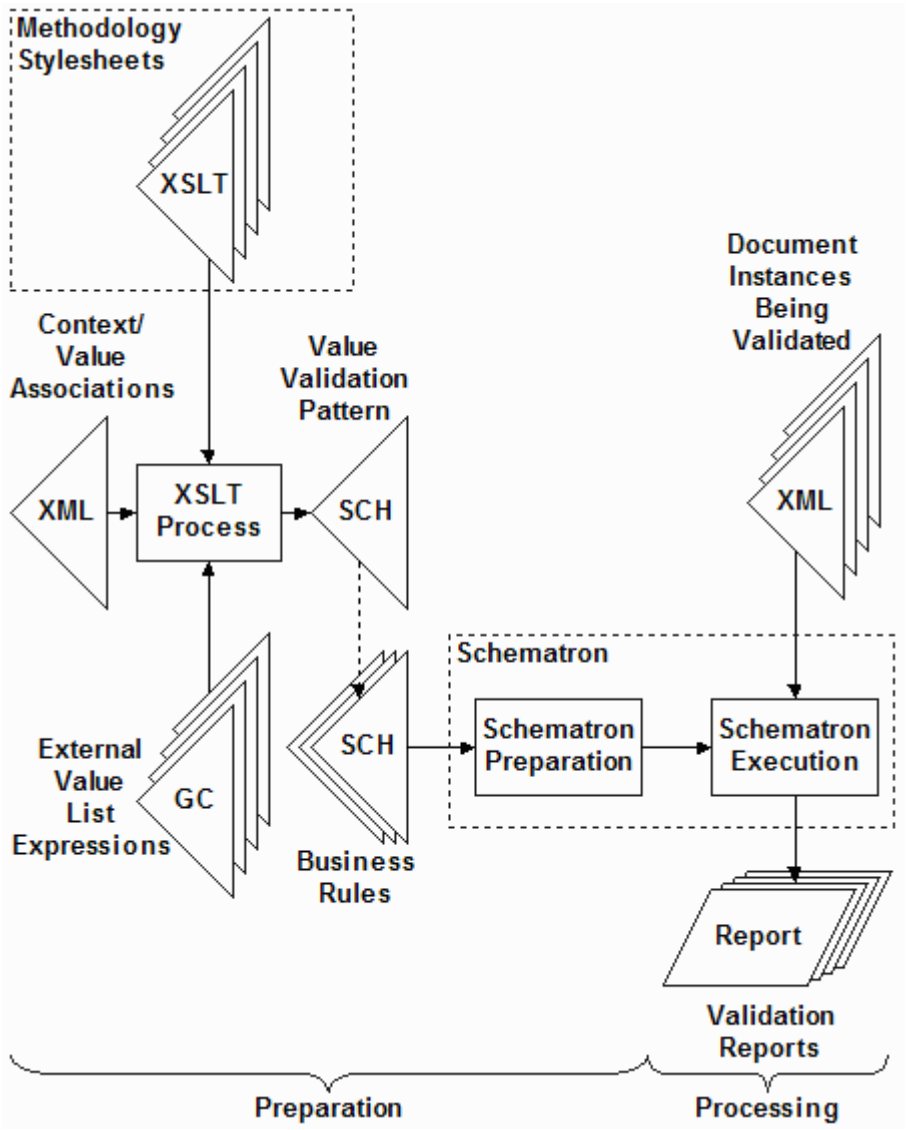
Figure 1. Methodology context association



ISO Schematron [Schematron] is an assertion-based schema language used to confirm the success or failure of a set of assertions made about XML document instances. One can use ISO Schematron to express assertions supporting business rules and other limitations of XML information items so as to aggregate sets of requirements for the value validation of documents.

The synthesis of a pattern of ISO Schematron assertions to validate the values found in document contexts, and the use of ISO Schematron to validate those assertions are illustrated in Figure 2, “Methodology overview”.

Figure 2. Methodology overview



To feed the ISO Schematron process, one needs to express the contexts of information items and the values used in those contexts. This methodology prescribes an XML vocabulary to create instances that express such associations of values for contexts. The stylesheets provided with this methodology read these instances of context/value associations that point to externally-expressed lists of values and produce an ISO Schematron *pattern* of assertions that can then be combined with other patterns for business rule assertions to aggregate all document value validation requirements into a single process.

The validation process is then used against documents to be validated, producing for each document a report of that document's failures of assertions.

4.2. Example context/value association file

The following complete example of a context/value association file constrained by this specification and supporting an example scenario described at Section 10, “A complete running example” incorporates all the features described in this specification (these features are specified in detail in various sections of this document and not here in this overview section):

```
<?xml version="1.0" encoding="US-ASCII"?>
<ValueListConstraints
  xmlns="urn:oasis:names:tc:ubl:schema:ValueList-Constraints-1.0"
```

```
  xmlns:cbc="urn:oasis:names:draft:ubl:schema:xsd:CommonBasicComponents-2"
  xmlns:cac="urn:oasis:names:draft:ubl:schema:xsd:CommonAggregateComponents-2"
  id="urn:x-illustration"
  name="code-list-rules">

  <Title>
    Illustration of code list constraints - order-constraints.xml
  </Title>

  <Identification>
    $Id: order-constraints.xml,v 1.16 2006/11/23 11:36:06 G. Ken Holman Exp $
  </Identification>

  <Description>
    This is an illustrative example of all of the features of specifying the
    context/value constraints that one can express for XML documents.

    The validation requirements for this contrived scenario are as follows:
      - the UN/CEFACT currency code list is restricted to be
        only Canadian and US dollars
      - the seller must be in the US
      - the buyer may be in either Canada or the US
      - the definition for Payment Means is extended to include
        both UBL definitions and additional definitions
  </Description>

  <!--list all of the genericode expressions of agreed-upon code list
      value enumerations-->
  <ValueLists>
    <ValueList xml:id="currency" uri="CAUS_CurrencyCode.gc">
      Restricted to only Canadian and US dollars.
      <MetaData>
        <Reference>CurrencyCode</Reference>
        <Name>Currency</Name>
        <ID>ISO 4217 Alpha</ID>
        <URI>urn:un:unece:uncefact:codelist:specification:54217</URI>
        <Version>2001</Version>
<VersionURI>urn:un:unece:uncefact:codelist:specification:54217:2001</VersionURI>
       <AgencyName>United Nations Economic Commission for Europe</AgencyName>
        <AgencyID>6</AgencyID>
      </MetaData>
    </ValueList>
    <ValueList xml:id="states" uri="US_CountrySubentityCode.gc">
      List of US states
    </ValueList>
    <ValueList xml:id="provinces" uri="CA_CountrySubentityCode.gc">
      List of Canadian provinces
    </ValueList>
    <ValueList xml:id="tax-ids" uri="TaxIdentifier.gc">
      List of tax type identifiers
    </ValueList>
    <ValueList xml:id="payments" uri="UBL_PaymentMeansCode-2.0.gc">
     Copied from the UBL 2.0 suite: http://docs.oasis-open.org/ubl/cs-UBL-2.0/
    </ValueList>
    <ValueList xml:id="additional_payments"
               uri="Additional_PaymentMeansCode.gc">
      An extra set of possible payment means.
    </ValueList>
```

```
  </ValueLists>
  <!--list all of the contexts in which the value enumerations are used;
      where two or more contexts might match a given node in the input,
      list them here in order of most-important to least important match-->
  <Contexts>
    <Context item="@currencyID" values="currency">
      All currencies are restricted to only Canadian and US dollars.
    </Context>
    <Context item="cbc:CountrySubentityCode"
             context="cac:BuyerCustomerParty"
             values="provinces states">
      The buyer can be in either Canada or the US.
    </Context>
    <Context item="cbc:CountrySubentityCode"
             context="cac:SellerSupplierParty"
             values="states">
      The seller can only be in the US.
    </Context>
    <Context item="cbc:ID" xpath="cac:TaxCategory/cbc:ID" values="tax-ids">
      Limit the recognized tax identifiers
    </Context>
    <Context item="cbc:PaymentMeansCode"
             values="payments additional_payments">
      The payments can be by either standard or supplemental means.
    </Context>
  </Contexts>
</ValueListConstraints>
```

5. Code list and value definitions

5.1. Code list and value definitions inside schemata

Some of the constraints expressed in a schema constrain the values used for an information item by specifying the lexical and value limitations used in an XML instance. Such constraints may enumerate all of the possible coded values in a code list, or a list of identifiers, or any other kind of value. Alternatively, the constraints may merely limit the value lexically to an expected pattern without limiting the actual values used that match that pattern (for example, a token string of characters without any embedded white space).

How such constraints are expressed in a schema impacts on the flexibility of trading partners to use subsets, supersets or simultaneously use different sets of values for a given value list in a given XML instance that needs to be validated by the schema.

5.1.1. Value list constraints with enumerations

Schema expressions constraining the values for an information item often use `xsd:enumeration` elements restricting a base data type of either a normalized string or a tokenized value. The UBL declaration for currency coded values imported from UN/CEFACT uses such an approach as illustrated by this incomplete fragment:

```
  <xsd:simpleType name="CurrencyCodeContentType">
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="AED">
        <xsd:annotation>
          <xsd:documentation>
            <ccts:CodeName>Dirham</ccts:CodeName>
            <ccts:CodeDescription></ccts:CodeDescription>
          </xsd:documentation>
```

```
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="AFN">
      <xsd:annotation>
        <xsd:documentation>
          <ccts:CodeName>Afghani</ccts:CodeName>
          <ccts:CodeDescription></ccts:CodeDescription>
        </xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="ALL">
      <xsd:annotation>
        <xsd:documentation>
          <ccts:CodeName>Lek</ccts:CodeName>
          <ccts:CodeDescription></ccts:CodeDescription>
        </xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="AMD">
    ...
```

Trading partners may wish to contractually constrain the set of values for these information items to only those that are allowed in instances being exchanged. This is easily accommodated in the information exchange since any value used from a strict subset is, by definition, a value in the full enumeration. Instances with one of the limited values can be validated by the unchanged schema for the document model with all of the values.

Note that while techniques are available in some schema expression languages for restricting a data type definition of an enumeration to a subset of values, such a restriction typically has global scope across the instance. Trading partners may wish to constrain the values to different sets in different contexts of the document, which is not possible by way of available restriction techniques in some schema language expression semantics.

Trading partners wishing to extended the set of values for these information items are unable to do so when the value list constraints in a schema are expressed with an enumeration. Adding a new value to the list changes the definition of the list such that instances with new values cannot be validated by the unchanged schema for the document model.

5.1.2. Value list constraints without enumerations

Some value lists are declared in schemata without an enumeration, as there may be far too many possible values to be manageable, the sets of values may differ in different contexts of a single document, or the values are not predefined in any way. Schema expressions constraining such values often merely constrain the value to a normalized string or a tokenized value. This satisfies the lexical requirements of the value without constraining the particular values that meet the requirements.

A UBL example of a code list with an unmanageable number of enumerations is `cbc:CountrySubentityCode` as there are so very many provinces, states, regions, prefectures and other administrative or jurisdictional areas in the world that would be included to be complete. UBL does not supply any sets of values to use.

A UBL example of a code list with predefined values that trading partners may wish to extend or restrict is `DocumentStatusCodeType` where the UBL committee has chosen a set of meaningful values for a certain class of workflow definitions, but trading partners may have a richer set employed in their respective systems. This predefined list, among other predefined lists, is supplied in UBL using an external code list expression conforming to this methodology.

An example of a code list without any predefined values is `cac:AccountTypeCode` as there may be as many account types as there are trading parties, and none are predefined by UBL. Trading partners wishing to validate information items using coded values from this code list are obliged to agree on and express the set of values they expect to use. UBL does not supply any sets of values to use.

UBL uses a common base type for all elements expressing values from code lists. This declaration is illustrated by this incomplete fragment showing the meta data for the code list:

```
<xsd:complexType name="CodeType">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         ...
      </xsd:documentation>
   </xsd:annotation>
   <xsd:simpleContent>
      <xsd:extension base="xsd:normalizedString">
         <xsd:attribute name="listID" type="xsd:normalizedString"
                        use="optional"/>
         <xsd:attribute name="listAgencyID" type="xsd:normalizedString"
                        use="optional"/>
         <xsd:attribute name="listAgencyName" type="xsd:string"
                        use="optional"/>
          <xsd:attribute name="listName" type="xsd:string" use="optional"/>
          <xsd:attribute name="listVersionID" type="xsd:normalizedString"
                        use="optional"/>
          <xsd:attribute name="name" type="xsd:string" use="optional"/>
          <xsd:attribute name="languageID" type="xsd:language"
                        use="optional"/>
          <xsd:attribute name="listURI" type="xsd:anyURI" use="optional"/>
          <xsd:attribute name="listSchemeURI" type="xsd:anyURI"
                        use="optional"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

UBL uses a common base type for all elements expressing values that are identifiers. This declaration is illustrated by this incomplete fragment showing the meta data for an identifier:

```
<xsd:complexType name="IdentifierType">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         ...
      </xsd:documentation>
   </xsd:annotation>
   <xsd:simpleContent>
      <xsd:extension base="xsd:normalizedString">
         <xsd:attribute name="schemeID" type="xsd:normalizedString"
                        use="optional"/>
          <xsd:attribute name="schemeName" type="xsd:string" use="optional"/>
           <xsd:attribute name="schemeAgencyID" type="xsd:normalizedString"
                        use="optional"/>
           <xsd:attribute name="schemeAgencyName" type="xsd:string"
                        use="optional"/>
          <xsd:attribute name="schemeVersionID" type="xsd:normalizedString"
                        use="optional"/>
           <xsd:attribute name="schemeDataURI" type="xsd:anyURI"
                        use="optional"/>
          <xsd:attribute name="schemeURI" type="xsd:anyURI" use="optional"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

Note in both of the above how the optional meta data attributes do not have fixed attributes, thus allowing the XML instance to indicate not only the value but also the meta data related to the value list from which the value is taken.

5.1.3. UBL schemata constraints on code lists

UBL includes both kinds of constraints (with and without enumerations) for coded values in code lists, very few of which are declared with enumerations inherited from the UN/CEFACT unqualified data types. The `CurrencyCodeContentType` data type is used for `udt:AmountType` and the list of values delivered in the package matches those internationally standardized [currency]. The only other two such code lists in UBL enumerate MIME encoding identifiers in `BinaryObjectMimeCodeContentType` for `udt:BinaryObjectType` and unit code values in `UnitCodeContentType` for `udt:MeasureType`.

All other code lists and value lists in UBL are not enumerated in the schema expressions. Some code lists have a set of predefined values supplied by UBL that trading partners may wish to extend. Other lists can only have values provided by the users, as standardization either hasn't happened or shouldn't happen because it isn't appropriate to set out what trading partners are allowed to use.

The UBL 2.0 package includes genericode files for every code list, but not for any identifiers or other value lists. All of the lists supporting the unqualified data types have complete sets of coded values distilled from the schema expressions. There are a limited number of code lists with values predefined by the UBL committee, and all other code lists have an empty set of values. Trading partners can modify or replace any of the genericode files to meet their business requirements using the code list and value validation methodology in this Working Draft.

5.2. Code list and value definitions outside schemata

At this time of writing there are no standardized approaches to formally publishing the enumeration of members of a code list that is published by a code list maintainer or any value list of identifiers or other agreed-upon values. Such a formal representation is required for the purposes of machine processing suitable to methodologies supported by automated processes.

The genericode [genericode] approach is a de facto implementation of such a formal expression cataloguing the members of an enumeration with associated member and list documentation and list meta data description. This approach is undergoing OASIS standardization in the Code List Representation Technical Committee. This approach is used in the illustration of this value validation methodology, and has been adopted by (and is recommended by) the Universal Business Language Technical Committee for use in UBL.

It is with genericode files that this methodology's included stylesheets support trading partners expressing the values agreed to being used in document contexts. Other external expressions of values in a code list or value list can be accommodated by one of at least the two following methods:

- rewrite this methodology's implementation's stylesheet fragment accessing genericode files with an equivalent fragment that accesses the alternative format;

- transliterating instances of the alternative format to be minimal instances of the genericode format and using this methodology's implementation's existing stylesheet fragment.

The examples used in this methodology documentation are created by hand for illustrative purposes, and indeed a number of the UBL genericode files were also hand-authored. Of course genericode files need not be created this way and could be synthesized as the result of a database query or some other process to create the XML expression of the values.

It is not, however, part of the genericode specification at this time to be able to identify, parameterize or utilize an algorithmic determination of a set of codes in real time. Such synthesis can be used out-of-band to create the genericode file, but the genericode file then merely contains the end result of any such synthesis.

5.2.1. Trading partner genericode definitions

Trading partners will need to specify the genericode definitions for those value lists they wish to restrict or for those they wish to extend. Their flexibility in extending the value list is based on whether the list is described in the schema with or without a schema enumeration, as the presence of a schema enumeration prohibits extending the value list.

When specifying the values for a new list described in the schema without an enumeration, trading partners can modify a copy of an existing genericode file or synthesize a new genericode file from scratch and include the values agreed to be used.

An excerpt from the `scenario/CA_CountrySubentityCode.gc` file created by hand to represent the list of Canadian provinces and territories after April 1, 1999 reads as:

```
<gc:CodeList xmlns:gc="http://genericode.org/2006/ns/CodeList/0.4/">
   <Identification>
      <ShortName>provinces</ShortName>
      <LongName>Canadian Provinces</LongName>
      <Version>2</Version>
      ...
   </Identification>
   ...
   <SimpleCodeList>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>AB</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>Alberta</SimpleValue>
         </Value>
      </Row>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>BC</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>British Columbia</SimpleValue>
         </Value>
      </Row>
      ...
```

An excerpt from the `scenario/US_CountrySubentityCode.gc` file created by hand to represent US states reads as:

```
<gc:CodeList xmlns:gc="http://genericode.org/2006/ns/CodeList/0.4/">
   <Identification>
      <ShortName>states</ShortName>
      <LongName>US States</LongName>
      <Version>1</Version>
      ...
   </Identification>
   ...
   <SimpleCodeList>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>AL</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>ALABAMA</SimpleValue>
         </Value>
      </Row>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>AK</SimpleValue>
         </Value>
         <Value ColumnRef="name">
```

```
            <SimpleValue>ALASKA</SimpleValue>
         </Value>
      </Row>
      ...
```

The UBL 2.0 package includes a meta-data-only genericode file for the country sub-entity code and either or both of the above lists could be used in place of this file for validation.

5.2.2. Normative UBL genericode specification and use

To satisfy the normative requirements for references in UBL, the UBL set of deliverables includes a snapshot of the genericode specification.

The UBL 2.0 package includes a complete set of genericode files conforming to this snapshot specification, with which trading partners can tailor their needs. Each of the 92 code lists in UBL schemata are accommodated by a genericode file structured as one of the following:

- a complete set of predefined coded values specified by UN/CEFACT and matching the enumeration of values in the UBL schemata; these sets of values can be restricted by trading partners to only a subset of the predefined values required for validation;
- a complete set of predefined coded values specified by the UBL Technical Committee and reflecting system design properties of UBL as a whole; these sets of values can be restricted or extended by trading partners to be any set required for validation;
- an empty set with no coded values; these sets can be replaced by trading partners to be any set required for validation.

In addition two special-purpose code lists are included in the UBL package for trading partners to choose to use but not used by default.

UBL 2.0 does not include any genericode files for identifiers or other values.

6. Meta data for information item values in XML instances

In the absence of meta data properties for values in the instance being validated, only the values found in the associated external list representation can be used. There being no qualification of the values in the instance, all values in the external file are in play as valid values for validation.

However, if the instance being validated does have meta data properties specified for a given value, then that value is asserted to be a value from a particular version or identified list of values. Therefore, when the set of values is qualified by meta data properties indicating information about the set itself, the validation process can accurately correlate the instance's value with a set's values.

An external XML representation of a set of values will typically have meta data. The context/value association file declaration of the external resource can masquerade these values with overriding values. This is very important when the external resource is a set of values derived from a larger set. Consider that a specification like UBL will specify that a country coded value is from a code list of the complete ISO set of country codes. UBL instances that specify meta data for a country value will specify the meta data of the complete list. When trading partners are using a derived list of country codes for validation, that list of codes necessarily has different meta data than the complete list of codes. To ensure that the validation of an instance using meta data to specify the meaning of the country code as being an ISO country code accepts the derived country code, the declaration of the derived list masquerades as the complete list while pointing to the derived list.

This methodology is supplied to function with information items in a UBL instance and the external XML representations of code lists in genericode instances. The methodology, however, works with information items in any

instance and any external XML representation of code lists, requiring only the included modular stylesheets to be modified in a plug-and-play method for their use in other validation scenarios.

6.1. UBL instance meta data

The UBL naming conventions for the meta data properties differ slightly based on the information item defined by a code list. The following summarizes the meta data attributes used:

- for an attribute matching `@currencyID`:
  - `@currencyCodeListVersionID`
- for an attribute matching `MeasureType/@unitCode`:
  - `@unitCodeListVersionID`
- for an attribute matching `QuantityType/@unitCode`:
  - `@unitCodeListID`, `@unitCodeListAgencyID` and `@unitCodeListAgencyName`
- for an element named `<xxxxxCode>` based on a code type:
  - `@listID`, `@listAgencyID`, `@listAgencyName`, `@listName`, `@listVersionID`, `@listURI` and `@listSchemeURI`
- for an element named `<yyyyyID>` based on an identifier type:
  - `@schemeAgencyID`, `@schemeAgencyName`, `@schemeName`, `@schemeVersionID`, `@schemeDataURI` and `@schemeURI`
- for all other attributes and elements defined by UBL
  - it is assumed there is no meta data that describes the value set from which an information item's values are constrained that is suitable for checking against the meta data found in genericode files
  - this does not prevent the code list value validation methodology from being used on these information items, only that there is no associated meta data about the set of values

Note that the context of an information item cannot be easily tested, so a simplification of the two contextual attributes above is implemented in the UBL code list validation meta data algorithm:

- for an attribute matching `@unitCode`:
  - `@unitCodeListVersionID`, `@unitCodeListID`, `@unitCodeListAgencyID` and `@unitCodeListAgencyName`

The above simplification will result in the validation checking for non-existent meta data attributes, but this is acceptable because the first-pass schema validation of a UBL instance will disallow the presence of the unexpected attributes. There is only a negligible overhead for checking all four meta data attributes when only one or three are present.

6.2. Context/value association specification of meta data

The UBL 2 code list meta data corresponds to the following meta data elements that are children of the `<Metadata>` element child of `<ValueList>` element in a context/value association file:

- there is no attribute that maps to the association `<Reference>`

- UBL `@listName` == association `<Name>`
- UBL `@listID` == association `<ID>`
- there is no attribute that maps to the association `<URI>`
- UBL `@listVersionID` == association `<Version>`
- UBL `@listSchemeURI` == association `<VersionURI>`
- UBL `@listURI` == association `<LocationURI>`
- UBL `@listAgencyName` == association `<AgencyName>`
- UBL `@listAgencyID` == association `<AgencyID>`

The UBL 2 identifier meta data corresponds to the following meta data elements that are children of the `<Meta-data>` element child of `<ValueList>` element in a context/value association file:

- there is no attribute that maps to the association `<Reference>`
- UBL `@schemeName` == association `<Name>`
- there is no attribute that maps to the association `<ID>`
- there is no attribute that maps to the association `<URI>`
- UBL `@schemeVersionID` == association `<Version>`
- UBL `@schemeURI` == association `<VersionURI>`
- UBL `@schemeDataURI` == association `<LocationURI>`
- UBL `@schemeAgencyName` == association `<AgencyName>`
- UBL `@schemeAgencyID` == association `<AgencyID>`

6.3. External value list meta data validation

When the context/value association file does not specify in a `<MetaData>` element a particular value of meta data for a value list, the external representation is examined for that meta data property to use for validation. This is examined separately for each meta data property, providing for selectively overriding values in the context/value association file and not having to specify all of them. Any externally-specified value for a particular meta data property is ignored when the association file specifies a value to be used for that property.

6.3.1. Genericode meta data and context/value association file meta data

Genericode has a number of child elements of `<Identification>` expressing meta data for the enumeration.

When genericode files are used for external expressions, the methodology will satisfy absent context/association meta data values as follows:

- association `<Reference>` == genericode `ShortName`
- association `<Name>` == genericode `LongName[1]`
- association `<ID>` == genericode `LongName[@Identifier='listID']` or `LongName[1]`

- association `<URI>` == genericode `CanonicalUri`
- association `<Version>` == genericode `Version`
- association `<VersionURI>` == genericode `CanonicalVersionUri`
- association `<LocationURI>` == genericode `LocationUri`
- association `<AgencyName>` == genericode `Agency/LongName`
- association `<AgencyID>` == genericode `Agency/Identifier`

7. Document contexts in XML instances

The different types of information items that are described by code lists and value lists are typically declared in few places in document models but, because of document context, the actual instances of these information items are found in possibly very many places in actual instances. Each use of an information item is in a different document context. Document grammars that validate information items based solely on global declarations cannot distinguish the uses of the items in different document contexts and any desired differences in value validation required by trading partners exchanging XML instances.

Document contexts are expressed structurally as hierarchical tree locations. Without confidence that the document contexts of the information items of an XML instance are sound, no amount of contextual checking of item values is going to be reliable. It is, therefore, a necessary precondition in advance of using this value validation methodology to validate the XML instances against a schema expression of structural constraints. This is a critically important step because the schema constraints will confirm the document contexts of information items are correctly positioned in the XML instance document hierarchy. Only when the information items are known to be in correct contexts will the value checking of the document contexts reflect bona fide results.

In UBL the XML document constraints are expressed using the W3C Schema [W3C Schema] language. This suffices to be a structural schema for all of the information items, and in addition describes the enumerations for the code lists for UN/CEFACT-based data types.

7.1. Using XPath to specify document context

The XML Path Language 1.0 [XPath 1.0] is used to address locations in an XML document according to a data model of processed syntax. This XPath data model differs from other data models such as the Document Object Model [DOM] in that the DOM models more aspects of raw syntax used in the document. Given that syntax is irrelevant (in that it is arbitrary to the creator of XML which syntactic choices are made when marking up documents) the XPath data model is sufficient to talk about the elements and attributes found in documents.

Elements are referred to in an XPath expression by their namespace-qualified names, while the "`@`" character (an abbreviation for the XPath `attribute::` axis) prefixes attributes referred to by their namespace-qualified names.

Note

XPath 1.0 considers names without prefixes to always be in no namespace, and never uses the default namespace to qualify names without prefixes. For this reason, all namespace-qualified information items in an XML vocabulary being validated must be prefixed when being addressed in XPath 1.0, even if the instances of this vocabulary utilize the default namespace.

The syntax of an XPath expression separates multiple location steps of a single location path using an oblique "`/`" character. Each step to the right names the child element or attached attribute of the immediately preceding step to the left which is always an element. Child elements are one level deeper in the XML hierarchical nesting than their parents. Elements are also parents of their attached attributes.

A fully-qualified absolute XPath location path begins with the oblique indicating the path starts from the root node (the parent of the document element) of the XPath data model document tree. A relative XPath location path starts with the name of an information item without the oblique at the beginning.

Examples of four absolute XPath location paths possible for an instance of UBL Order are as follows (note that arbitrary white-space is allowed between steps of an XPath address):

```
/po:Order/cac:TaxTotal/cbc:TaxAmount/@currencyID
/po:Order/cbc:DocumentCurrencyCode
/po:Order/cac:BuyerCustomerParty/cac:Party/cac:PostalAddress/
                                          cbc:CountrySubentityCode
/po:Order/cac:SellerSupplierParty/cac:Party/cac:PostalAddress/
                                          cbc:CountrySubentityCode
```

An example of a relative XPath location that matches all currency values in attributes in the entire instance of any UBL document model is as follows, as the information item does not include any ancestral distinction to the left:

```
@currencyID
```

An example of a relative XPath location that matches all country sub-entity values in elements in the entire instance of any UBL document model is as follows, as the information item does not include any ancestral distinction to the left:

```
cbc:CountrySubentityCode
```

The minimum XPath addresses needed to precisely distinguish the country sub-entity code of the party address of each of the buyer and seller are as follows, as the information item includes explicit ancestry to the left:

```
cac:BuyerCustomerParty/cac:Party/cac:PostalAddress/cbc:CountrySubentityCode
cac:SellerSupplierParty/cac:Party/cac:PostalAddress/cbc:CountrySubentityCode
```

Note the use of the "`//`" operator in XPath allows the matching within an entire sub-tree of the hierarchy; the XPath addresses needed to distinguish all (not just in the party address) country sub-entity codes descendent to the buyer and the seller would be as follows indicating only the required (and possibly distant) ancestor:

```
cac:BuyerCustomerParty//cbc:CountrySubentityCode
cac:SellerSupplierParty//cbc:CountrySubentityCode
```

7.2. Trading partner uses of document contexts

When deciding on code list and value validation, trading partners must agree in which contexts particular sets of values need to be constrained.

Some business rules may require the same context to be specified across all document types, such as "All currency values must be Canadian or US dollars."

Other business rules may require indistinct document contexts to be specified, such as "all country sub-entity codes used in the order and in the invoice shall be valid states according to the United States postal service."

Note

In fact constraining only the country sub-entity code could be quite misleading in a business environment, but it is used here for illustrative purposes. The code "`WA`" validly representing both Western Australia, Australia and Washington, United States of America, would be an example of the ambiguity problem.

Yet other business rules might require more distinct document contexts to be specified, such as "The country sub-entity codes for the seller can only be states of the United States, while country sub-entity codes for the buyer can be both provinces of Canada and states of the United States."

Furthermore, trading partners can choose to employ an agreed-upon controlled vocabulary for document contexts for which values are not defined. This UBL value validation methodology is agnostic to the method by which

information items are declared in schemata, thus allowing trading partners to specify acceptable values for information items in any context.

Trading partners must, therefore, take the step to agree on which XPath addresses will specify the contexts at which particular values are constrained. Examining the list of contexts in which constrained-value information items are found, the partners can identify as much specificity as is required to match those contexts in which the values are constrained.

7.3. UBL reporting of document contexts of code list information items

The UBL support package includes code list document context reports for every document type of the UBL suite. Each context report lists all of the minimally-unique document contexts for information items based on code lists in that document type definition expressed by the schema. These reports are algorithmically derived from the UBL W3C Schema expressions.

The number of code-list-based information items ranges from a low of 14 found in the Order Response Simple model, to a high of 77 found in the Freight Invoice model.

The number of minimally-unique code-list-based information item document contexts ranges from a low of 311 found in the Attached Document model, to a high of 153,335 found in the Order Response model.

8. Specifying value constraints in document contexts

This Working Draft describes the document model for a context/value association file in which associations are made between document contexts of information items and external resources expressing the values allowed for those items.

This model is found in the compressed package associated with this specification in the `utility/` directory in the `UBL-ContextConstraints-0.8.xsd` file. This model needs an external declaration of the `xml:id=` attribute, for which one is supplied named `xmlid.xsd` (derived from the xml:id Recommendation [xml:id]).

Using this document model, trading partners create an instance of context/value associations. The instance points to genericode files as system resources using URI strings, and names these pointers using XML identifiers unique to the instance. The document context of each information item to be validated using this methodology is then associated with as many pointer identifiers as required to enumerate all of the possible values from all of the possible enumerations.

A pro-forma context/value association instance reads as follows:

```
<?xml version="1.0" encoding="US-ASCII"?>
<ValueListConstraints
  ... namespace declarations as required for XPath addresses ...
  xmlns="urn:oasis:names:tc:ubl:schema:Value-List-Constraints-1.0"
  id="urn:x-optional-unique-identifier-for-external-referencing"
  name="required-unique-name-token-for-internal-referencing">

  <Title>
    This is the main context/value association file for project X.
  </Title>

  <Identification>
    Revision: 27b - 2006-11-23 15:00z
  </Identification>
```

```
  <Description>
    This is included for illustrative purposes.
  </Description>

  <Include uri="other-assoc-file-1.xml">
    Incorporating information from another file.
  </Include>
  <Include uri="other-assoc-file-2.xml">
    Incorporating information from yet another file.
  </Include>

  <ValueLists>
    <ValueList xml:id="a1" uri="enumeration1.gc">
      A set of external values with no masquerading meta data.
    </ValueList>
    <ValueList xml:id="a2" uri="enumeration2.gc">
      A set of external values with masquerading version meta data.
      <MetaData>
        <Version>1.4</Version>
      </MetaData>
    </ValueList>
    <ValueList xml:id="a3" uri="enumeration3.gc">
      Another set of external values with no masquerading meta data.
    </ValueList>
  </ValueLists>

  <Contexts>
    <Context item="@item-a" values="a2">
      Associating a set of values with all item-a= attributes.
    </Context>
    <Context item="item-b" context="context-b1" values="a1 a3">
      Associating two sets of values with all item-b descendants
      of the context-b1 element.
    </Context>
    <Context item="item-b" xpath="context-b2/item-b" values="a3">
      Associating a set of values with item-b children of the
      context-b2 element.
    </Context>
  </Contexts>
</ValueListConstraints>
```

The required `name=` attribute specifies a name token for internal referencing by downstream processes that take advantage of alternative expressions of this information.

The optional `id=` attribute specifies a public identifier (typically, but not required to be, a URI) for external referencing, such as in business process specifications and formal trading partner agreements.

The optional `<Title>` element is for reporting purposes and is typically a single line of text suitable for the titling of the report. Its string value (stripping markup) is copied to the resulting validation file to indicate where portions of its information originated when there are a number of portions included. This element can have any content for documentation purposes, provided elements in that content are in a foreign namespace and not in the context/value association file vocabulary.

The optional `<Identification>` element's string value (stripping markup) is also copied to the resulting validation file to indicate where portions of its information originated when there are a number of portions included. This element can have any content for documentation purposes, provided elements in that content are in a foreign namespace and not in the context/value association file vocabulary.

The optional <Description> element is typically a prose description of the association file and its use. This element can have any content for documentation purposes, provided elements in that content are in a foreign namespace and not in the context/value association file vocabulary.

The optional and repeatable <Include> element is a directive to incorporate the associations found in other context/value association files into the one generated result. Where two contexts from the suite of association files match the same node, the priority for the single match that is acted upon is highest for the contexts in the invoked association file, then next highest for the last association file included by an <Include> directive (e.g. those in other-assoc-file-2.xml above), then the next-to-last association file included by an <Include> directive (e.g. those in other-assoc-file-1.xml above), and so on with the lowest priority being the first association file included by an <Include> directive. Any included association files having such directives will treat those in priority before other directives of the including file. This element can have any content for documentation purposes, provided elements in that content are in a foreign namespace and not in the context/value association file vocabulary.

Each <ValueList> element child of <ValueLists> declares the unique identifier for the external expression of values and the pointer to the associated system resource itself, in this example a genericode-encoded file. This element can have any content for documentation purposes, provided elements in that content are in a foreign namespace and not in the context/value association file vocabulary.

Each <Context> element child of <Contexts> points to all of the lists of values for a given information item in the values= attribute as a white-space-separated list of <ValueList> identifiers. This element can have any content for documentation purposes, provided elements in that content are in a foreign namespace and not in the context/value association file vocabulary.

This element must declare the XPath address of the information item in isolation without context using the item= attribute (the first example above is an attribute, the other two are elements). This is sufficient contextual information when testing the item in a document-wide context, however, when the information item needs to be tested in a given sub-document context or a subset of a multi-document context, one of two mutually-exclusive attributes is required.

The context= attribute specifies some ancestral element of the information item beneath which all information items addressed are to be considered in context. Alternatively, when more nuanced detail is required, the xpath= attribute specifies the precise XPath context of the information item, including in the address the information item itself. These two attributes can be any valid XPath expression, with as much context and as many predicates as is needed to identify the constructs in the instances.

Note

The equivalence of contextual XPath addresses documented above can be expressed that for a given item="a", having context="b" is equivalent to having xpath="b//a" in its place. Validation error reports will report the equivalent xpath= value, whether specified or not, to ensure an unambiguous report.

It is possible that regardless of which combination of attributes is used, two XPath expressions will both match the same node in the source document. The <Context> elements for the context/value associations must, therefore, be ordered with the more important XPath contexts first and the less important XPath contexts following in order to ensure a predictable result. A given information item from the document being validated can match only a single context declaration in this context/value association file.

Trading partners can choose to have separate expressions of context/value associations for instances of each document type, or using the context of the document element, combine the associations in a single file for a subset of a multi-document context. An example of a context/value association file testing information items in both UBL Order and UBL Invoice instances is as follows, where @item-a has the same constraints in both instances, but @item-b has different constraints in both instances:

```
<?xml version="1.0" encoding="US-ASCII"?>
<ValueListConstraints
```

```
  xmlns:in="urn:oasis:names:specification:ubl:schema:xsd:Invoice-1.0"
  xmlns:po="urn:oasis:names:specification:ubl:schema:xsd:Order-1.0"
  xmlns="urn:oasis:names:tc:ubl:schema:Value-List-Constraints-1.0"
  name="code-list-rules">
  <ValueLists>
    <ValueList xml:id="a1" uri="enumeration1.gc"/>
    <ValueList xml:id="a2" uri="enumeration2.gc"/>
    <ValueList xml:id="a3" uri="enumeration3.gc"/>
  </ValueLists>
  <Contexts>
    <Context item="@item-a" values="a2"/>
    <Context item="@item-b" context="/po:Order//context-b1"
             values="a1 a3"/>
    <Context item="@item-b" xpath="/in:Invoice//context-b1/@item-b"
             values="a3"/>
  </Contexts>
</ValueListConstraints>
```

8.1. Choosing between `context=` and `xpath=` attributes

Section 4.2, “Example context/value association file” illustrates the following three uses of the `<Context>` attributes named `context=` and `xpath=` (the documentation has been elided):

```
    <Context item="@currencyID" values="currency"/>
    <Context item="cbc:CountrySubentityCode"
             context="cac:BuyerCustomerParty"
             values="provinces states"/>
   <Context item="cbc:ID" xpath="cac:TaxCategory/cbc:ID" values="tax-ids"/>
```

The first declaration does not use these attributes, thus specifying the constraints for all uses of the `currencyID=` attribute across the entire document instance. This is the most typical use of the `<Context>` element as most requirements constrain information items in all places where the item is used.

The second declaration uses `context=` to specify the constraints for all `<cbc:CountrySubentityCode>` elements that are descendants (including immediate children) of the `<cac:BuyerCustomerParty>` element. This context is reported in error messages as the XPath string `cac:BuyerCustomerParty//cbc:CountrySubentityCode`. This is the next most typical use of the `<Context>` element as there may be some requirements for constraining values in only parts of a document.

The third declaration uses `xpath=` to specify a precise XPath address constraining the `cbc:ID` element only when it is an immediate child of the `cac:TaxCategory` element. Rarely needed, the `xpath=` attribute must be used when using `context=` creates a scope too broad for accurate testing and reporting. Consider the following snippet of a UBL 2.0 instance where inappropriately using `context="cac:TaxCategory"` would unnecessarily (and likely in error) validate `cac:TaxScheme/cbc:ID` with the same constraints as `cac:TaxCategory/cbc:ID`:

```
<cac:TaxSubTotal>
  <cbc:TaxableAmount currencyID="USD">100.00</cbc:TaxableAmount>
  <cbc:TaxAmount currencyID="USD">15.00</cbc:TaxAmount>
  <cac:TaxCategory>
    <cbc:ID>Z</cbc:ID>
    <cbc:Percent>15</cbc:Percent>
    <cac:TaxScheme>
      <cbc:ID>Provincial Tax</cbc:ID>
    </cac:TaxScheme>
```

```
  </cac:TaxCategory>
</cac:TaxSubTotal>
```

8.2. Restricting an existing code list

When limiting the values from a list described in the schema with an enumeration, the genericode file supplied in UBL is initialized to include all enumerated values. Trading partners can then work from this complete list and prune unwanted values in a copy leaving in the values agreed to be used in XML instances. Changing the list, however, obligates the user to change the meta data of the list, as the new list does not in fact reflect the complete list as described by the original meta data.

An example of this is the file `cl/gc/cefact/CurrencyCode-2.0.gc` based on the UN/CEFACT currency values and including over 160 entries. A copy of this file named `scenario/CAUS_CurrencyCode.gc` is edited where the entire list of coded values has been pruned to only the Canadian dollar and the US dollar, and the meta data necessarily modified to reflect the qualified list and not the complete list. The file reads as follows:

```
<gc:CodeList xmlns:gc="http://genericode.org/2006/ns/CodeList/0.4/">
   <Identification>
      <ShortName>CAUSCurrencyCode</ShortName>
      <LongName>Canadian and US Currency Codes</LongName>
      <Version>1</Version>
      <CanonicalUri>urn:x-company:CAUS-currency</CanonicalUri>
     <CanonicalVersionUri>urn:x-company:CAUS-currency:1</CanonicalVersionUri>
   </Identification>
   <ColumnSet>
      <Column Id="code" Use="required">
         <ShortName>Code</ShortName>
         <Data Type="xsd:normalizedString"/>
      </Column>
      <Column Id="name" Use="optional">
         <ShortName>Name</ShortName>
         <Data Type="xsd:string"/>
      </Column>
      <Key Id="codeKey">
         <ShortName>CodeKey</ShortName>
         <ColumnRef Ref="code"/>
      </Key>
   </ColumnSet>
   <SimpleCodeList>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>CAD</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>Canadian Dollar</SimpleValue>
         </Value>
      </Row>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>USD</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>US Dollar</SimpleValue>
         </Value>
      </Row>
   </SimpleCodeList>
</gc:CodeList>
```

Note that such an edited list of values cannot be used in the validation of a UBL instance without accommodating the presence of its unique, non-UBL meta data. Consider the following snippet of a UBL 2.0 instance that specifies version meta data for the currency attribute:

```
<cac:TaxTotal>
  <cbc:TaxAmount currencyCodeListVersionID="2001"
                 currencyID="USD">15.00</cbc:TaxAmount>
  <cbc:TaxEvidenceIndicator>false</cbc:TaxEvidenceIndicator>
  <cac:TaxSubTotal>
```

This instance validates with the supplied UBL 2.0 pass-two validation because the meta data matches that found in the genericode file supplied with UBL 2.0. This instance would not, however, validate with the restricted code list above because the version in the restricted code list, amongst other meta data, doesn't match UBL 2.0 meta data.

The context/value association file in Section 4.2, “Example context/value association file” accommodates this with the following declaration of the use of the restricted code list, while masquerading the restricted list of values as being the bona fide complete UBL list of values:

```
<ValueList xml:id="currency" uri="CAUS_CurrencyCode.gc">
  Restricted to only Canadian and US dollars.
  <MetaData>
    <Reference>CurrencyCode</Reference>
    <Name>Currency</Name>
    <ID>ISO 4217 Alpha</ID>
    <URI>urn:un:unece:uncefact:codelist:specification:54217</URI>
    <Version>2001</Version>
<VersionURI>urn:un:unece:uncefact:codelist:specification:54217:2001</VersionURI>
    <AgencyName>United Nations Economic Commission for Europe</AgencyName>
    <AgencyID>6</AgencyID>
  </MetaData>
</ValueList>
```

By masquerading the meta data, the code list genericode file properly includes the unique meta data of the derived set of codes, while the validation properly matches against UBL 2.0 meta data values of the list from which the derived set is obtained. There is, of course, a risk that one could accidentally or maliciously improperly purport a derived list to be from a different list and cause instances to pass value validation without an error.

8.3. Extending an existing code list

The intuition to extend a code list merely by adding new code list items causes problems with the integrity of the meta data and validation of the list values. Changing a list of codes necessarily requires the use of different code list meta data than that used for the original list. If instances being validated are using meta data from the original list, there will be a mismatch of meta data on a code that is defined in the original list.

To properly extend the set of values with which to validate an information item, one must independently declare the extended set of codes from the base set of codes, and then associate the information item with both lists of codes. This fragmentation is arbitrary and one could have any number of sets of codes, each with their own meta data, and associate an information item with all of them.

Consider the declaration of the code lists and context/value association for the `<cbc:PaymentMeansCode>` in the complete file in Section 4.2, “Example context/value association file”:

```
    <ValueList xml:id="payments" uri="UBL_PaymentMeansCode-2.0.gc">
     Copied from the UBL 2.0 suite: http://docs.oasis-open.org/ubl/cs-UBL-2.0/
    </ValueList>
    <ValueList xml:id="additional_payments"
               uri="Additional_PaymentMeansCode.gc">
      An extra set of possible payment means.
```

```
  </ValueList>
  ...
  <Context item="cbc:PaymentMeansCode"
           values="payments additional_payments">
    The payments can be by either standard or supplemental means.
  </Context>
```

A UBL instance can now specify payment means in a number of various methods. Consider first when meta data is not being used ... both of these would validate, the first from UBL and the second from the additional set of codes:

```
<cac:PaymentMeans>
  <cbc:PaymentMeansCode>1</cbc:PaymentMeansCode>
</cac:PaymentMeans>
```

```
<cac:PaymentMeans>
  <cbc:PaymentMeansCode>SHP</cbc:PaymentMeansCode>
</cac:PaymentMeans>
```

When qualified with the appropriate meta data for the list from which the value is obtained, these two examples would also work without error:

```
<cac:PaymentMeans>
  <cbc:PaymentMeansCode listID="UN/ECE 4461">1</cbc:PaymentMeansCode>
</cac:PaymentMeans>
```

```
<cac:PaymentMeans>
  <cbc:PaymentMeansCode
     listID="Additional Payment Means">SHP</cbc:PaymentMeansCode>
</cac:PaymentMeans>
```

Finally, when qualified with the inappropriate meta data for the list from which the value is obtained, the following example would appropriately throw a validation error using an extended value in a UBL list, as the value "SHP" is not in the UN/ECE list:

```
<cac:PaymentMeans>
  <cbc:PaymentMeansCode listID="UN/ECE 4461">SHP</cbc:PaymentMeansCode>
</cac:PaymentMeans>
```

8.4. Documenting context/value association files

The context/value association file in Section 4.2, “Example context/value association file” includes a number of simple text strings to document for the human reader the information found in the file. In fact the documentation could include rich markup such as HTML or DocBook and need not only be simple text. Such markup is obliged to be in a foreign namespace, that is, it must use a namespace other than that of the context/value association vocabulary.

The following context/value association vocabulary constructs allow for embedded text and rich foreign-namespace markup:

- `<Title>`
  - typically a short piece of information used in a report to identify the file regardless of the version, copied into intermediate files to identify the file used as a source
- `<Identification>`
  - typically a short piece of information, also copied into intermediate files to unambiguously identify a version of the file used as a source (perhaps through the use of a time stamp)

  - this information is for the human reader and is distinct from the optional `<ValueListConstraints id="...uri...">` attribute for machine processing and identification of the file for business process specification or other formal uses.

- `<Description>`
  - a prose description typically used as an overview of the file and its use
- `<Include>`
  - prose information documenting the incorporation of an external definition of constructs
- `<ValueList>`
  - documenting the external list of values being declared

    Note

    The allowance of foreign content can result in mixed content for the `<ValueList>` construct when the documentation is comprised of simple text and not rich markup. This text is, in effect, mixed with the presence of the optional `<MetaData>` element (which itself must only be an immediate child of `<ValueList>`). This is not an error, but the processing of this text may require special attention in some tools.

- `<Context>`
  - documenting the contextual use of lists of values

Text and foreign-namespace content is not allowed at any other part of a context/value association file.

9. Value validation

The validation process involves checking all of the information items of an XML instance for their being in the context/value associations, and if so, having the instance values and their associated meta data checked against the values and meta data in the corresponding external XML representations of the value lists.

9.1. Using ISO/IEC 19757-3 Schematron

The ISO assertion-based schema language Schematron Section 9.1, “Using ISO/IEC 19757-3 Schematron” works by validating the information items in an instance against a set of assertions. A Schematron validating process tests each information item against the highest-priority assertion (earliest in the list of assertions) to which its XPath address matches.

The compressed package that is part of this methodology includes the `schematron-ISO-assembly.xsl` and `schematron-ISO-incomplete-text.xsl` XSLT 1.0 stylesheets (in the `utility/` subdirectory) that are used in tandem to transform a compound set of Schematron sets of assertions into a single XSLT 1.0 stylesheet that, when applied against a document to be validated, reports any failed assertions that are detected. All validation failures are sent to the operating system standard error port, and a non-zero exit is returned from the XSLT processor. A zero exit returned from the XSLT processor indicates successful validation.

Note

The supplied `schematron-ISO-assembly.xsl` and `schematron-ISO-incomplete-text.xsl` XSLT 1.0 stylesheets are incomplete implementations of ISO Schematron in anticipation of later receipt of complete and conforming implementations. The versions included in the test scenario are modified Schematron 1.5 stylesheets, changed to recognize the ISO namespace in order that Schematron expressions written by early adopters of this methodology will be portable to complete XSLT implementations of ISO Schematron when available. Other implementations of ISO Schematron are publicly available (e.g. Scimitar [Scimitar]).

This methodology implements code list and value validation by expressing the assertions that the information items in the instance being validated, with any related meta data, are valid values from the external code list expressions that are associated to the information items through the context/value association files. This is accomplished in a modular fashion so as to mesh with other business rules that trading partners may need to express regarding their agreed-upon electronic documents.

9.2. Methodology XSLT transformation stylesheets

The compressed package that is part of this methodology includes an XSLT 1.0 stylesheet (complete with imported fragments) that transforms a context/value association file for a UBL document model and genericode external value list expressions into a corresponding Schematron set of assertions in a single named Schematron pattern.

These stylesheets are modular in a fashion that allows new stylesheets to take advantage of existing modules to support the methodology with other document models and other external code list expressions.

When adapting this methodology to document models and external value list expressions, the basic stylesheet filename patterns follow these conventions:

- `{documentModel}-{externalFormat}2Schematron.xsl` is the stylesheet being invoked,
- `{documentModel}-Metadata.xsl` is the module identifying the meta data in the instance being validated,
- `{externalFormat}-CodeList.xsl` as the module identifying the meta data in the external code list expression, and
- `Constraints2Schematron.xsl` is the module directing the creation of the resulting `Schematron` expression.

Following this convention, then, the demonstrative example included with this methodology invokes the following stylesheet and fragments:

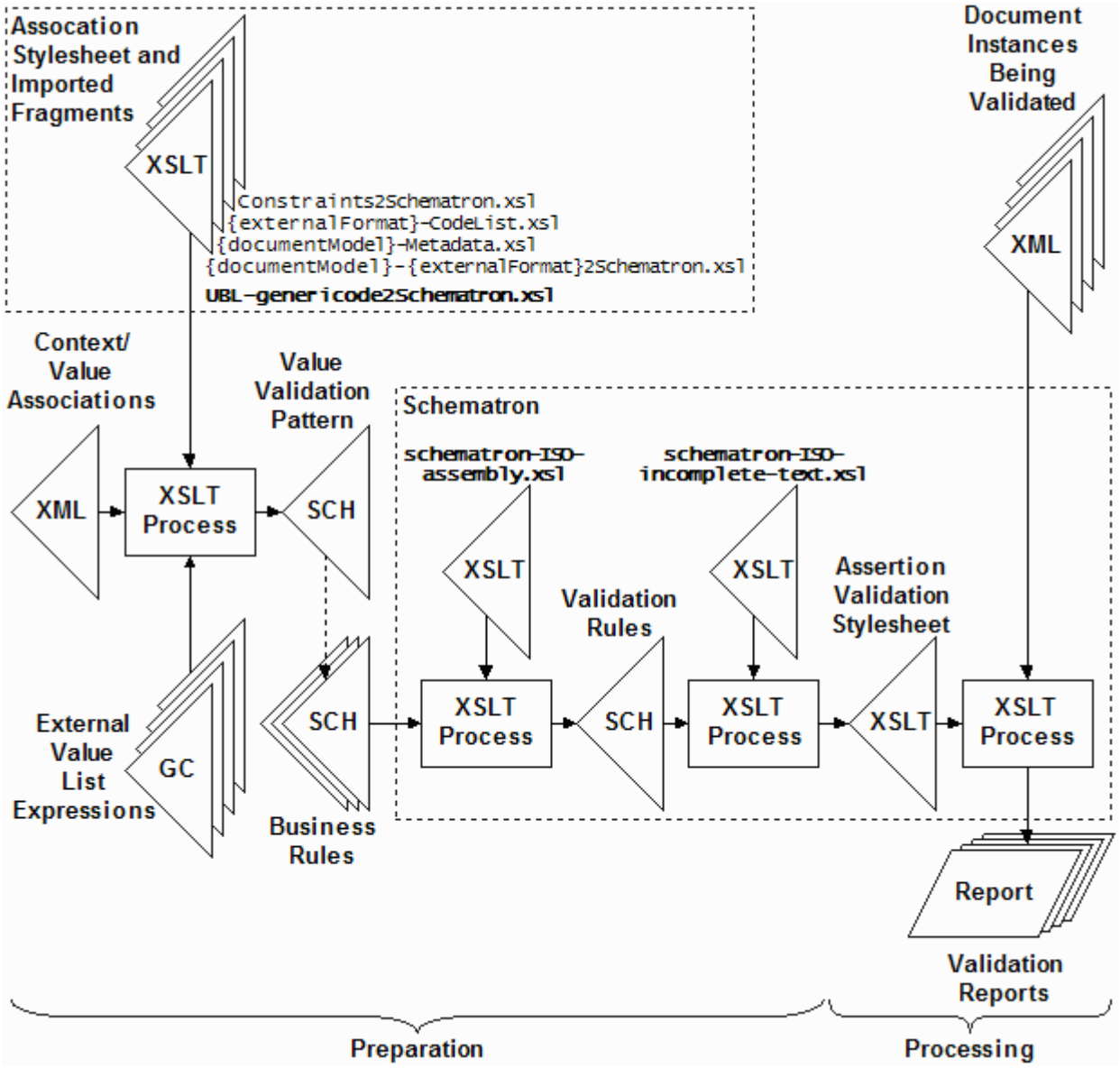
- `UBL-genericode2Schematron-0.8.xsl` which imports the other fragments,
- `UBL-metadata-0.8.xsl`,
- `genericode-CodeList-0.8.xsl` and
- `Constraints2Schematron-0.8.xsl`.

The `Constraints2Schematron-0.8.xsl` stylesheet creates a Schematron pattern, named using the `name=` attribute in the context association file, in the standalone output file. Each file's `<Context>` element becomes a Schematron assertion (thus requiring the document order of `<Context>` elements to be the required priority order). This end result can then be incorporated into complete Schematron schemata by reference using the `<sch:include>` directive that is resolved in a strict assembly stage that takes place in advance of semantic interpretation of the other Schematron constructs

9.3. Methodology data flow diagram

The demonstrative example implementation of this methodology has two distinct phases, one for the preparation of the validation process and one for the validation activity itself, as shown in Figure 3, “Methodology data flow”. The information expressed in the context/value association file is thus manipulated, but only when the inputs change, into a form with which the actual validation of the document instances is accomplished as many times as required.

Figure 3. Methodology data flow



There is no obligation that an implementation of this methodology follow this particular data flow, only that the end result include the same reported validation violations of contextual use of coded values in the document instances to be validated.

Note

The inputs to the Schematron dotted box in Figure 3, “Methodology data flow” can only be Schematron schemas. How Schematron is implemented is not important to this methodology. The flow described inside of the Schematron dotted box is the flow implemented by the XSLT stylesheets in the ZIP package supplied with this methodology. The XSLT implementation provides for a runtime component being only the execution of an XSLT stylesheet. Other implementations of Schematron may use other runtime components.

9.3.1. Validation preparation

The preparation phase incorporates a number of steps to interpret the requirements for validation in order to prepare the validation artefact.

The context/value association file points each of the document instance contexts to the associated external code list expressions. The XSLT stylesheet that imports fragments with knowledge of the document model's meta data and the external code list expression structures transforms the input information into a set of Schematron assertions that need to be true about the document being validated. These assertions are exported in a Schematron file with only a single named pattern of code list rules.

An including Schematron schema of business rules uses `<sch:include>` to incorporate the code list rules with possibly other files of business rules. The Schematron assembly step assimilates all included constructs into a complete Schematron schema of all validation rules.

The resulting Schematron schema is then interpreted into an assertion validation XSLT stylesheet, which is that artefact that implements the checking of the assertions against an instance for validity.

9.3.2. Validation processing

The XSLT expression of Schematron assertions created in the preparation phase is run against all of the document instances being validated to produce the corresponding validation reports.

There is no need to recreate the assertion validation stylesheet unless anything changes in the context/value associations and external code list expressions, which must then be reprocessed to create a replacement validation artefact.

9.4. Necessary preconditions for the methodology

Not illustrated in Figure 3, “Methodology data flow” are three necessary preconditions for the data flow to produce bona fide validation reports. The three inputs to the data flow must each be validated against their respective document models in advance to provide properly structured information to the internal processes. None of the XSLT processes illustrated perform any validation of the inputs.

The context/value associations file must have been validated against the `UBL-ContextConstraints-0.8.xsd` constraints.

The external code list expressions must have been validated against their respective model, in this example the `genericode-code-list-0.4.xsd` constraints.

The intermediate file labeled "Validation Rules" in the diagram can be validated against the supplied `ISOSchematron.rnc` file. Note that at this time there is no published schema with which to validate standalone Schematron patterns and business rules that are not assembled into a complete Schematron schema.

The documents being validated with this methodology must have been validated against their respective structural schema model, which is not included in this example. This is a critically important precondition because the schema constraints will confirm the information items are correctly positioned in the XML instance document hierarchy. This ensures the XPath instructions in the context/value association will be properly applied. Without having confirmed the structural integrity of the XML instance, the assertion validation report is meaningless.

10. A complete running example

The compressed package that is part of this methodology includes the `scenario/` subdirectory in which the following complete scenario can be run to demonstrate how a file of context/value associations can be used to validate sample UBL instances using this methodology and the documented data flow.

Not included in this demonstration is the necessary step run in advance of the code list value validation methodology of validating the UBL document instances against the UBL W3C Schema expression of structural constraints. As noted above, this precondition ensures the information items of the instances are correctly placed in the document hierarchy to be tested for their validity, thus producing bona fide validation reports.

10.1. Support files required

In both the Linux shell environment and the Windows command-line environment, two shell scripts or batch files are invoked in the test scenario and must be available on the path. The following illustrates how each of these are invoked, the first line for a Windows environment and the second line for a Linux environment:

- ```
  call w3cschema schema-file instance-file
  sh w3cschema.sh schema-file instance-file
  ```

  This invokes a W3C Schema validating processor applying the given W3C Schema set of constraints against the given instance file. A non-zero error is returned if there are any validation errors.

- ```
  call xslt input-file stylesheet-file output-file
  sh xslt.sh input-file stylesheet-file output-file
  ```

  This invokes an XSLT stylesheet processor applying the given stylesheet file against the given input file to produce the named output file. Optionally, any number of name/value pairs can be supplied to bind values to top-level parameters in the stylesheet.

10.1.1. Engaging the illustrative example

The invocation files in the ZIP package are preconfigured ready to use once a number of required JAR files are copied into the scenario directory.

The supplied Java-based command-line invocation of the Xerces [Xerces] W3C Schema processor is xjparse [xjparse], invoked as follows (with the appropriate classpath set):

- `java com.nwalsh.parsers.xjparse -S schema-file instance-file`

The supplied Java-based XSLT processor is Saxon [Saxon], invoked as follows (with the appropriate classpath set):

- `java -jar saxon.jar o output-file input-file stylesheet-file param=value`

The classpaths supplied assume the following JAR files are copied into the `utility/` directory before use:

- from xerces: `resolver.jar` and `xercesImpl.jar`
- from xjparse: `xjparse.jar` (copied from a versioned filename)
- from Saxon 6.5.5: `saxon.jar`

Of course the supplied invocation files can be replaced with invocation files of your choice.

10.2. Scenario

In this scenario two trading partners are going to interchange UBL documents between buyer and seller parties. At a technical level, they are using unmodified UBL W3C Schema expressions publicly available for the structural integrity of their XML instances.

At a business level, the trading partners have agreed that all currencies used in an instance can be only Canadian or US dollars. The `CAUS_CurrencyCode.gc` file documented in Section 5.2.1, “Trading partner genericode definitions” expresses this limited number of coded values.

The partners have also agreed that the buyer's country sub-entity codes may be either a US state or a Canadian province, but that the seller's country sub-entity codes may only be a US state. The two genericode files `US_CountrySubentityCode.gc` and `CA_CountrySubentityCode.gc`, also documented in Section 5.2.1, “Trading partner genericode definitions”, express these limitations.

Using the `TaxIdentifier.gc` set of identifiers, the transactions can make reference to appropriate taxes.

Finally, the trading partners have agreed to use both the complete set of payment means used in UBL 2.0 plus an additional payment means not used in UBL 2.0, expressed in `Additional_PaymentMeansCode.gc`.

The context/value association file `order-constraints.xml` is listed in its entirety in Section 4.2, “Example context/value association file” and points each of the UBL instance contexts to the required genericode files in order to satisfy the trading partner agreement for this scenario.

These genericode files and context/value association file together form a formal and unambiguous expression of the contextual coded value constraints that go beyond the constraints of the standardized UBL schema expressions. The package of these files can, therefore, be included in a contractual agreement between the trading partners.

The artefact in this scenario produced by the stylesheet is the file `order-constraints.sch` which includes only a single named Schematron pattern, suitable for inclusion in any ISO Schematron schema. The following Schematron schema `codes-only-constraints.sch` is a minimum expression suitable for including the generated named pattern, as it is acceptable that an assembled Schematron schema contain only a single pattern:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <title>Code list value assertions</title>
  <ns prefix="cbc"
      uri="urn:oasis:names:draft:ubl:schema:xsd:CommonBasicComponents-2"/>
  <ns prefix="cac"
     uri="urn:oasis:names:draft:ubl:schema:xsd:CommonAggregateComponents-2"/>
  <include href="order-constraints.sch"/>
</schema>
```

Moreover, trading partners may have additional business rules that apply to instances. Consider there might also be a Schematron expression limiting the total amount of the invoice to less than $10,000. This business rule would be in its own pattern, as in the example `total-limits-constraint.sch` schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<pattern xmlns="http://purl.oclc.org/dsdl/schematron" id="total-limit">
  <rule context="cbc:ToBePaidAmount">
    <assert test=". &lt; 10000">Total amount '<value-of select="."/>'
cannot be $10,000 or more</assert>
  </rule>
</pattern>
```

The Schematron file including the synthesized pattern and the authored pattern could be expressed merely as two patterns in the schema without any phases, thus implying that all patterns are in play at all times. Alternatively, for more validation flexibility if desired, this can be expressed in Schematron semantics as a single phase that would indicate both patterns as active in the validation process using a Schematron schema along the following lines:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
        defaultPhase="only-phase">
  <title>Business rules for maximum total value</title>
  <ns prefix="cbc"
      uri="urn:oasis:names:draft:ubl:schema:xsd:CommonBasicComponents-2"/>
  <ns prefix="cac"
     uri="urn:oasis:names:draft:ubl:schema:xsd:CommonAggregateComponents-2"/>
  <phase id="only-phase">
    <active pattern="code-list-rules"/>
    <active pattern="total-limit"/>
  </phase>
  <include href="total-limit-constraint.sch"/>
  <include href="order-constraints.sch"/>
</schema>
```

Such business rules could also form part of the contract between trading partners, and for one of the tests in the scenario the generated Schematron rules augment authored Schematron rules to make a single assertion validation expression for use against the instances.

10.3. Running test instances against the scenario

A number of test instances are included to demonstrate the code list value validation methodology in this test scenario:

- `order-test-good1.xml` and `order-test-good2.xml` are two instances with valid coded values in context for currency and country sub-entity codes; the buyer in the first instance is in the US and the buyer in the second instance is in Canada; note that the Canadian address uses meta data to indicate version 2 of the associated code list (it happens that a new Canadian territory named Nunavut was recently created by the splitting of the Northwest Territories, thus increasing the number of country sub-entity codes after many decades of not having changed);

- `order-test-bad1.xml` uses the schema-valid currency coded value "UYU" for Peso Uruguayo (which happens to be next in the code list to the US dollar, and might have been inadvertently selected in a data entry user interface), but this violates the trading partner agreement to use only US or Canadian dollars, while it would have not violated the UBL W3C Schema expression;

- `order-test-bad2.xml` uses the coded value for Canadian country sub-entity code with the meta data for the list indicating the coded value is from the old version "1" of the code list, but this violates the trading partner agreement to use only codes from version "2" of the code list; note that while the same coded value happens to be used as in version "1", it cannot be assumed to be valid because the coded value might represent different semantics in version "2" and the trading partners have agreed to use the updated version;

- `order-test-bad3.xml` uses a Canadian province for the country sub-entity code of the seller's address, but this violates the trading partner agreement that the seller must have a US address. This instance also has a typographical error in the amount to be paid, having omitted the decimal separator, thus indicating a total of $11,500 and not $115.00.

First the scenario establishes the preconditions by validating the context/value association expression of constraints in `order-constraints.xml` and each of the genericode code list expressions (the necessary W3C Schema validation of the test input files is not included in the demonstration).

Next, the code list rules expressed in the context/value association file are translated into a Schematron pattern in `order-constraints.sch`. This can now be reused wherever code list constraints need to be checked. Note that the comments created in this pattern file include the markup for namespace declarations that are necessary in the including Schematron schemas. It is the user's responsibility to ensure that all of the namespaces required by all of the included Schematron pattern files are appropriately declared in the including schema.

The first test scenario, named "Test 1", translates these inputs without business rules into Schematron expression in `codes-only-constraints.sch` without any supplemental business rules, and then translates that first into a complete Schematron instance order-`codes-only.sch` and from there into an XSLT 1.0 expression in `order-codes-only.xsl`.

Preparation is complete for the first test and now the validation stage runs this resulting XSLT against all the test files, producing no errors and a zero return code when there are no problems, and producing a list of errors and a non-zero return code when there are problems. The error report is output to the standard error port, and the return code is testable by the script running the process.

The second test scenario, named "Test 2", augments the Schematron expression of business rules in `total-constraints.sch` with the code list rules into `order-codes-total.sch` and then translates that into an XSLT 1.0 expression in `order-codes-total.xsl`.

Preparation is complete for the second test and the validation stage runs the resulting XSLT against the `order-test-bad3.xml` file with the corrupted total amount, indicating the violation of both the code list and business constraints.

Note again how the constraints need only be prepared once, translating the requirements into the XSLT in the preparation phase to the artefact which is then reused during the validation phase as often as required. In a production environment the XSLT used for validation need only be recreated whenever the trading partner agreement changes to include a new formal expression of the coded value constraints in either the context/value association file or a code list expression file.

Run "`sh test-all.sh`" in a Linux environment or "`test-all.bat`" in a Windows command-line environment to get the following results of running the test scenario:

```
Precondition validation...

Validating partner-agreed constraints...
  w3cschema UBL-ContextConstraints-0.8.xsd order-constraints.xml
Attempting validating, namespace-aware parse
Parse succeeded (0.241) with no errors and no warnings.

Validating code lists...
  w3cschema genericode-code-list-0.4.xsd CA_CountrySubentityCode.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.291) with no errors and no warnings.
  w3cschema genericode-code-list-0.4.xsd US_CountrySubentityCode.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.301) with no errors and no warnings.
  w3cschema genericode-code-list-0.4.xsd CAUS_CurrencyCode.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.270) with no errors and no warnings.
  w3cschema genericode-code-list-0.4.xsd TaxIdentifier.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.270) with no errors and no warnings.
  w3cschema genericode-code-list-0.4.xsd Additional_PaymentMeansCode.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.270) with no errors and no warnings.

Preparing code list rules...

Translating partner-agreed constraints into Schematron rules...
  xslt order-constraints.xml ..\utility\UBL-genericode2Schematron-0.8.xsl
                                                    order-constraints.sch

Test 1 - standalone code list rules

Assembling rules into a Schematron schema...
  xslt codes-only-constraints.sch ..\utility\schematron-ISO-assembly.xsl
                                                    order-codes-only.sch

Translating Schematron into validation stylesheet...
  xslt order-codes-only.sch ..\utility\schematron-ISO-incomplete-text.xsl
                                                    order-codes-only.xsl

Document validation...

Testing order-test-good1.xml...
  xslt order-test-good1.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 0


Testing order-test-good2.xml...
  xslt order-test-good2.xml order-codes-only.xsl nul "2>test-constraints.txt"
```

```
Result: 0


Testing order-test-bad1.xml...
  xslt order-test-bad1.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 1
Value supplied ' UYU ' is unacceptable for values identified by 'currency'
in the context '@currencyID': /Order/cac:TaxTotal/cbc:TaxAmount/@currencyID

Processing terminated by xsl:message at line 21


Testing order-test-bad2.xml...
  xslt order-test-bad2.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 1
Value supplied ' ON ' is unacceptable for values identified by 'provinces
states' in the context 'cac:BuyerCustomerParty//cbc:CountrySubentityCode':
/Order/cac:BuyerCustomerParty/cac:Party/cac:Address/cbc:CountrySubentityCode

Processing terminated by xsl:message at line 21


Testing order-test-bad3.xml...
  xslt order-test-bad3.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 1
Value supplied ' ON ' is unacceptable for values identified by 'states' in
the context 'cac:SellerSupplierParty//cbc:CountrySubentityCode':
/Order/cac:SellerSupplierParty/cac:Party/cac:Address/cbc:CountrySubentityCode

Processing terminated by xsl:message at line 21


Test 2 - with business rules

Assembling rules into a Schematron schema...
  xslt total-constraints.sch ..\utility\schematron-ISO-assembly.xsl
                                                    order-codes-total.xsl

Translating Schematron into validation stylesheet...
  xslt order-codes-total.sch ..\utility\schematron-ISO-incomplete-text.xsl
                                                    order-codes-total.xsl

Document validation...

Testing order-test-bad3.xml...
  xslt order-test-bad3.xml order-codes-total.xsl nul "2>test-constraints.txt"
Result: 1
Total amount ' 11500 ' cannot be $10,000 or more:
/Order/cac:LegalTotal/cbc:ToBePaidAmount
Value supplied ' ON ' is unacceptable for values identified by 'states' in
the context 'cac:SellerSupplierParty//cbc:CountrySubentityCode':
/Order/cac:SellerSupplierParty/cac:Party/cac:Address/cbc:CountrySubentityCode

Processing terminated by xsl:message at line 22


Done.
```

11. Future work

The UBL TC has decided to postpone publishing version 1.0 of this methodology until genericode 1.0 has been standardized and published. At that point all of the examples and supporting stylesheets will be modified to support the standardized representation of a list of coded values.

The compressed package that is part of this methodology includes a modified version of Schematron 1.5, changed to recognize the ISO Schematron namespace but not implementing the ISO Schematron functionality. ISO Schematron has been standardized and when a proper implementation of it is created the test scenario will be changed to employ the latest freely-available version. Nevertheless, the provided incomplete stylesheet is sufficiently functional to implement the checking of a wide range of business rules.

References

[currency] UN/ECE Working Party on Facilitation of International Trade Procedures *Alphabetical Code for the Representation of Currencies [http://www.unece.org/cefact/recommendations/rec09/rec09_ecetrd203.pdf]* Recommendation 9 (Second Edition) January 1996, ECE/TRADE/203 [Edition 96.1]

[DOM] World Wide Web Consortium *Document Object Model [http://www.w3.org/DOM/DOMTR]*

[genericode] Tony Coates *genericode [http://www.genericode.org/], UBL Code List Representation Technical Committee [http://www.oasis-open.org/committees/codelist/]*

[ISO 3166-2] *International Organization for Standardization ISO 3166-2:1998 Codes for the representation of names of countries and their subdivisions - Part 2: Country subdivision code [http://www.iso.org/iso/en/prods-services/iso3166ma/04background-on-iso-3166/iso3166-2.html]*

[OASIS] *Organization for the Advancement of Structured Information Standards [http://www.oasis-open.org/]*

[RFC Keywords] S. Bradner *Key words for use in RFCs to Indicate Requirement Levels [http://rfc.net/rfc2119.txt]* Internet Engineering Task Force, March 1997

[Saxon] Michael Kay *Saxon [http://saxon.sf.net/]*

[Schematron] Rick Jelliffe *ISO/IEC 19757-3 Schematron [http://www.schematron.com/], Document Schema Definition Languages (DSDL) Part 3 [http://www.DSDL.org], ISO/IEC JTC 1/SC 34/WG 1 [http://www.jtc1sc34.org/]*

[Scimitar] Fourthought, Inc. *Scimitar [http://uche.ogbuji.net/tech/4suite/amara/]*

[UBL 2.0] Jon Bosak, Tim McGrath, G. Ken Holman *Universal Business Language (UBL) Version 2.0 [http://docs.oasis-open.org/ubl/os-ubl-2.0/], OASIS UBL Technical Committee [http://www.oasis-open.org/committees/ubl/]* 2006

[UBL 2.0 Support] Jon Bosak *Universal Business Language (UBL) Version 1.0 Support Materials [http://docs.oasis-open.org/ubl/cd-UBL-1.0/]* (currently under revision), *OASIS UBL Technical Committee [http://www.oasis-open.org/committees/ubl/]* 2005-09-15

[UBL 1.0 SBS] *Universal Business Language (UBL) Version 1.0 Small Business Subset [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl-sbsc]*

[W3C Schema] *XML Schema Part 0: Primer [http://www.w3.org/TR/xmlschema-0/], XML Schema Part 1: Structures [http://www.w3.org/TR/xmlschema-1/], XML Schema Part 2: Datatypes [http://www.w3.org/TR/xmlschema-2/]* 2004-10-28

[Xerces] The Apache XML Project *Xerces [http://xerces.apache.org/xerces2-j/]*

[xjparse] Norman Walsh *xjparse [http://nwalsh.com/java/xjparse/]*

[xml:id] Jonathan Marsh, Daniel Veillard, Norman Walsh *xml:id Version 1.0 [http://www.w3.org/TR/2005/REC-xml-id-20050909/], Annex D.2 [http://www.w3.org/TR/2005/REC-xml-id-20050909/#with-schema-validation]* 2005-09-09

[XPath 1.0] James Clark, Steve DeRose *XML Path Language (XPath) Version 1.0 [http://www.w3.org/TR/1999/REC-xpath-19991116]* 1999-11-16